User Guide to Anetd 1.6.9

Steve Dawson* Fred Gilham* Marco Molteni* Livio Ricciulli† Sonia Tsui*

June 28, 2001

*SRI International, System Design Laboratory
†Metanetwoks, Inc.

Contents

1 Introduction

Anetd is an experimental daemon specifically designed to support the deployment, operation and control of active networks [3]. The Anetd design philosophy is to be fully backward compatible with current networking software, and at the same time to introduce new innovative functionality in managing, designing and controlling active networks. Anetd doesn't need superuser privileges to run and has minimal installation requirements.

Today, Anetd can be be used to manage legacy networking software and more experimental software derived from active networking research. Anetd performs two major functions:

1. **Deployement**. It allows the deployment, configuration and control of networking software, including current active networking *execution environment* (EE) prototypes, into the network.

2. **Demultiplexing**. It demultiplexes active network packets, encapsulated using ANEP [1], to multiple EEs located on the same network node and sharing the same input port.

Anetd requires little or no change to either legacy networking software or current active networking prototypes and greatly improves the ability to manage large active networks.

1.1 Deployment Capability

Anetd allows different administrative domains to share the same software base specified as URLs, and allows the deployment and control of distributed network services through a centralized source of electronic knowledge. The network services to be deployed are specified as URLs; Anetd, after downloading a service with an HTTP GET command, strips the HTML header from the received code and installs the service.

Anetd supports the deployment of native binary compatible code or Java applications. The deployment and configuration mechanism is fully backward compatible. Legacy services can be configured by downloading configuration files and specifying required command-line arguments so that existing software can be easily deployed without modification.

In addition to legacy services, Anetd can be used to deploy and control ANCORS engineering services. For these applications, the daemon offers a set of built-in primitives, which are services that, in addition to standard native system functionality (I/O, memory management, networking), provide

1. multithreading (nonpreemptive)
2. LAN multicast emulation
3. global time synchronization

These primitives can provide support for distributed simulation network engineering applications, as well as some forms of sophisticated network monitoring. For more information, see [2].

1.2 ANEP Demultiplexing

Active networking is motivated by the notion that the improvement and evolution of current networking software is greatly hindered by slow and expensive standardization processes. Active networking tries to facilitate changes to the networking software by developing an architecture that, by design, allows safe and efficient dynamic reconfiguration of the network.

Anetd listens on a unique UDP port (e.g., the active networks port 3322) and accepts commands to download, configure and manage networking software. In addition to processing commands directed to itself, Anetd looks at the packets it receives and, if a packet is destined to one of the services it downloaded, it forwards that packet to the appropriate service. This kind of functionality is very useful in cases where several network services need to share a limited set of port numbers (as in the current architecture of an active node) and it is not possible or desirable to modify the lower level-protocols.

[Here we should put a drawing showing how the demultiplexing works]

1.3 Deployable Networking Applications

Anetd has been designed to require minimal modifications to traditional networking applications in order to deploy them.

The only general requirement is that the applications must be fairly encapsulated and do not have many dependencies (e.g. do not require a large number of specific shared libraries that are not commonly available). Anetd supports the deployment of resources needed by the applications to facilitate the porting of existing software, but Anetd is not intended to be used to deploy a large number of libraries or large portions of installation directories. In these cases, it is advisable to use standard manual installation methodologies or bundle the required resources with Anetd in advance.

Anetd demultiplexing uses the standard input of the deployed service for forwarding packets to it. In other words, as explained below, an application wanting to receive traffic demultiplexed by Anetd coming from an ANEP port must read its input stream from standard input. This is accomplished as follows: once a new application with an ANEP type assigned to it is deployed, Anetd adds the type ID to a demultiplexing table. The demultiplexing table maps an ANEP type

ID to a file descriptor; the file descriptor, in turn, maps to the standard input of the application. Application output streams do not require any support from Anetd and can allow the system to assign output ports automatically.[1]

2 Implementation

Anetd is written in the C programming language and, because it uses a standard Unix API, it should be easily portable to any Unix platform. At the time of this writing, Anetd is available for FreeBSD on x86, Linux on x86 and Solaris on Sparc.

[This section remains to be written]

3 Usage

Anetd listens on TCP and UDP ports (default 3322) and accepts ANEP [1] encapsulated packets. If the type ID in the ANEP packet header is of the type assigned to Anetd by ANANA (type 51), the ANEP payload is parsed and Anetd control commands are executed. A client application, called *sc*, allows the user to send commands to Anetd. The commands are signed using public key cryptography and are encapsulated in ANEP packets. Although the *sc* application embeds its commands in ANEP packets, it uses as default transport protocol TCP, to obtain automatic retransmission of lost packets.

[Here we should put a picture describing the various forwarding and demultiplexing actions that take place depending on the ANEP id.]

3.1 Anetd and the ABone

Anetd can run standalone or as part of an ABone node configuration. An ABone node configuration consists of several Anetd processes, each running in a different operating system (OS) user account (which must never be *root*). Typically, the set of user accounts used in a given ABone node configuration is {`anpub`, `abocc`, `anee1`, `anee2`, `anee3`, `anee4`, `anee5`}, corresponding to a set of ABone roles, administered by the ABone Coordination Center (ABOCC), to which users of the ABone (EE developers) are assigned according to their EE development projects.

As of Anetd 1.6.6 these ABone roles are no longer necessarily tied to OS accounts having the same names (although it is strongly recommended that any deviation from standard practice be coordinated with ABOCC). Instead, an Anetd process may run under any OS account (distinct from the OS account of any other Anetd process) and act in one of the standard ABone roles. This makes it possible to run multiple distinct ABone node configurations on a single host, where each node configuration uses a different and distinct set of OS accounts. At the same time, the Anetd processes in each node configuration are addressed in the standard way according to their respective ABone roles, so that each node configuration appears to the ABone as if it were configured in the standard way. For example, the Anetd processes for an ABone node might be configured to run under the OS accounts `acct1`, `acct2`, `acct3`, `acct4`, `acct5`, `acct6`, and `acct7`, but to act in the ABone roles `anpub`, `abocc`, `anee1`, `anee2`, `anee3`, `anee4`, and `anee5`, respectively. From the point

[1] As far as we understand, this demultiplexing scheme is analogous to the mechanism that is used in inetd.

of view of ABone users, all the standard ABone "accounts" exist on this node and can be addressed in the usual way. They need not even be aware that the Anetd processes are actually running under OS accounts with different names.

The remainder of this document uses the phrases "ABone accounts" and "ABone roles" interchangeably to refer to the standard set of roles in which Anetd processes act on the ABone. Where reference needs to be made to the underlying OS accounts, the term "OS account" is used.

Standalone Mode For local use and testing, it is sometimes convenient to run Anetd in *standalone* mode. Running Anetd in standalone mode changes the default values of certain configuration settings (see Section 3.4) and also disables the automatic downloading of access control files from ABOCC servers (see Section 3.2.3).

Invoking Anetd with the `-s` command-line switch explicitly forces Anetd to run in standalone mode. Anetd can also be implicitly put into standalone mode by running Anetd from an OS account that does not correspond to one of the ABone roles and *not* invoking it with the `-a` option to map it to one of those roles.

The following table attempts to make this clearer:

Command-line Argument	Example OS Account	Mode
-s	*any*	standalone mode
neither -s nor -a	user1234	standalone mode
neither -s nor -a	anpub	ABone mode
-a anpub	user1234	ABone mode

For more information on the ABone and ABone node configuration, see the main ABone web site: `http://www.isi.edu/abone/`. See Sections 3.4 and 3.5 of this document for information on Anetd configuration and invocation.

3.2 Access Control

Anetd, starting from version 1.0, employs 512-bit public key cryptography to authenticate control commands. Each control command sent to an Anetd daemon is digitally signed, thus ensuring that Anetd access control policies are soundly enforced.

Anetd enforces access control in two ways:

1. It executes deployment and control commands only originating from a set of known (client id, public key) pairs, specified in one or more *master* ACL (Access Control List) file(s) and/or one or more *local* ACL file(s)

2. It downloads and executes code only from a set of HTTP servers or local files/directories specified in one or more master TCL (Trusted Codeserver List) file(s) and/or one or more local TCL file(s)

3.2.1 ACL Format

Each ACL file is a list of entries, one per line, of the following form:

```
<Client ID> <public key> [P] [<ABone account>]
```

- **<Client ID>** is a string, used by Anetd as an index to the corresponding public key. One client can have as many client IDs as he wants, each associated with a public key[2]. However, on the ABone each principal must use his or her e-mail address as client ID for the public key registered with the ABOCC.
- **<public key>** is the client's 512-bit public key in printable ASCII. Any incoming request from the client `Client ID` will be verified using this key. In case a command fails the verification, Anetd returns the error `CLIENT_NOT_AUTHORIZED`.
- **P** indicates that the client is authorized to issue *permanent* commands (see section 3.8.1).
- **<ABone account>** In case this field is not present, any request coming from the client <Client ID> that is successfully verified will execute in the Anetd that received the request. If this field is present, the primary Anetd daemon will attempt to forward the incoming request to a daemon acting in the ABone role <ABone account>. See also section 3.3.2.

3.2.2 TCL Format

Each TCL file is a list of entries of the following form:

```
entry {
  servers {
    <URL_1>
       ⋮
    <URL_n>
  }
}
```

where each $<URL_i>$ is a URL of one of the following forms:

- `http://`$<host>$`[:`$<port>$`]/`$<path>$
- `file://`[`localhost`]`/`$<path>$

Each URL specifies either a single file or the root of a directory tree from which it is permissible to download code (via a LOAD command) to be executed by Anetd. A list of URLs in a single TCL entry is interpreted as a list of alternate (mirror) servers from which Anetd may try to download code in the event that one of them is inaccessible. HTTP URLs normally refer to remote hosts (often managed by the ABOCC), while file URLs refer to the host on which Anetd is running.

Anetd 1.6 will also read legacy TCL files, where each entry is simply the hostname of a code server, but their use is deprecated. Support for legacy TCL files will be removed in a future version of Anetd.

[2]Until version 1.3, the client IP address was used to find the corresponding public key.

3.2.3 ACL/TCL File Handling

The handling of ACL/TCL files by Anetd depends on whether Anetd is running as an ABone node or standalone.

If Anetd is running as an ABone node (see also section 3.1), it will first try to fetch from an ABOCC server the current *master* ACL/TCL files[3] for the ABone account under which it is run, replacing the existing master ACL/TCL files stored locally on the node. If Anetd fails to fetch any master ACL/TCL file from all configured ABOCC servers, it will use its existing local copies of the master ACL/TCL files, if present. After downloading, Anetd reads first all available master ACL/TCL files, followed by all available local ACL/TCL files.

If Anetd is running standalone, by default no master ACL or TCL file is fetched from the net, and Anetd reads first all locally available master ACL/TCL files, followed by all available local ACL/TCL files.

Whether or not Anetd is running in an ABone configuration, if no *ACL* files (master or local) are available Anetd exits with an error.

3.3 Forwarding

When multiple instances of Anetd are running (under different OS accounts), only one such instance may be listening on the default Anetd port (3322) (for any one network interface), and that instance (if it exists) is the *primary* Anetd instance. Other instances are considered *secondary*. To execute a command on a secondary Anetd, either the client must send the command directly to the port on which the secondary is listening, or the primary must forward the command to the secondary. Forwarding can be done either explicitly or implicitly in Anetd version 1.4 or higher.

Forwarding to secondary daemons increases response time. In order to improve performance, the daemon responding to a client's request includes the port number on which it is listening in the acknowledgement message. This allows a client to learn the port number after the first request and subsequently use that port to directly communicate with the secondary daemon without the additional forwarding delay. This port-mapping feature can be used in those cases in which performance is critical but can be ignored for normal operation.

Example 3.1
(see section 3.3.1 for the meaning of the `-f` switch):

```
$ sc -f abocc d04 QUERY
text/ancors: abocc@d04.csl.sri.com:3322->8000 DEVEL bsd44
anetd_uptime 1:01:25:34

$ sc d04:8000 QUERY
text/ancors: abocc@d04.csl.sri.com:3322->8000 DEVEL bsd44
anetd_uptime 1:01:25:48
```

In the first sc invocation we ask the primary Anetd on node d04 to forward our command, QUERY, to the secondary Anetd running in the ABone role `abocc`. The reply we get comes from that

[3] The default names and locations of ACL/TCL files are documented in Section 3.4.

secondary Anetd. Among other things we see the port number the secondary is listening, port 8000.

In the second sc invocation we send the QUERY command directly on port 8000, obtaining exaclty the same reply.

3.3.1 Explicit Forwarding

Although the explicit forwarding mechanism applies to any system that runs multiple instances of Anetd, the standard ABone configuration is used here to illustrate its operation. If a node has been properly configured for the ABone, it will have several Anetd processes, each running in a different OS account and acting in one of the following ABone roles: *anpub*, *anee1* .. *anee5*, *abocc*. Of those, only one (by default, *anpub*) will listen on the well-known port 3322.

To send a command to a secondary Anetd running under a specific ABone account, the client can send the command to the primary and request, via the `-f` switch, that it be forwarded to the secondary. For example, to send a QUERY command to the secondary Anetd running as *anee3*, the client can issue the following command:

```
sc -f anee3 <host> QUERY
```

The primary Anetd will simply forward the packet to the Anetd running under the ABone role *anee3*, without checking its ACL, since it is acting as a forwarder. The *anee3* Anetd will then check its ACL and act accordingly.

3.3.2 Implicit Forwarding

In implicit forwarding, the ACL of the primary Anetd controls where commands received by the primary[4] are forwarded. For example, suppose the ACL of a primary Anetd contains the following entry:

`user@myaddress.net` <public-key> `anee1`

and suppose a client issues the following command to that primary Anetd:

`sc` <host> `QUERY`

The command will be received by the primary Anetd (listening on port 3322). If the ACL entry listed above is the first matching entry for the client who issued the command, the primary will forward the command to the secondary Anetd running as *anee1*.

3.4 Anetd Configuration

Starting with version 1.6, most aspects of Anetd's behavior are controlled at runtime via configuration files. Anetd 1.6 supports the use of two configuration files: `anetd.conf` (master) and `anetd.conf.local` (local). The master configuration file, if present, is read first, followed by the

[4] Secondary Anetd instances do not perform forwarding.

local configuration file. The use of two configuration files is intended primarily for the ABone, where the ABOCC controls the master configuration file, while node administrators can make customizations in the local configuration file. A complete list of configuration items (keys) and their default values appears in Table 1.

The default location for the master configuration file depends on whether Anetd is configured for ABone operation (see section 3.1). If so, it looks for its master configuration file in

```
~<abocc_user>/anetd/etc/<abone_role>/anetd.conf
```

where <abocc_user> is the *OS username* of the Anetd process acting in the *abocc* role (default: abocc), and <abone_role> is the ABone role under which Anetd is running. If Anetd is running standalone, it looks for its master configuration file in

```
~/anetd/etc/anetd.conf
```

The location of the master configuration file can be set via the Anetd command line option -c <file-path> (see section 3.5). The default location for the local configuration file (in all cases) is ~/anetd/etc/anetd.conf.local, which can be modified by setting the configuration key local_conf_file in the *master* configuration file.

3.4.1 Configuration File Format

A configuration file is a list of whitespace separated entries of the form

```
<key> [=] <value>[, <value>, ...]
```

Integer and string values are distinguished automatically, except that values enclosed in quotes (") are interpreted as strings. String values containing any of the characters ",", "#", or "=" must be enclosed in quotes. Quoted strings may include C-style escape sequences.

String values may include substitutions of the form $(<key>), where <key> is the name of a defined string-valued key. For example, the two entries

```
this_key     "Hello"
that_key     "$(this_key) world."
```

would assign that_key the value "Hello world." Key substitution can be suppressed by escaping the dollar sign (i.e., \\$).

Tilde expansion (substitution of home directory for login name) is attempted on all string values whose first character is "~".

Comments begin with the pound character (#) and continue to the end of the line.

Multiple entries with the same key are equivalent to a single entry with the values of the multiple entries concatenated in a list. For example, the two entries

```
this_key     value1, value2
this_key     value3
```

are equivalent to the single entry

```
this_key     value1, value2, value3
```

This makes it possible, for example, for a local configuration file to add values to keys set in the main configuration file. Note, however, that values lists are not necessarily meaningful for all keys. Certain keys (as indicated in Table 1) are defined internally by Anetd and cannot have their values set or modified by configuration files.

3.4.2 Java Configuration

As of version 1.4 of Anetd, Java virtual machines (JVMs) are no longer bundled with the Anetd distribution. There are two main reasons for this:

1. The Anetd distribution becomes far too large with Java bundled, especially if more than one version of the JVM is needed.
2. Remote Java debugging is supported in version 1.4 and later of Anetd. Java debugging support requires the full Java Development Kit (JDK), not just the JVM, and the JDK license terms do not permit bundling with other software.

Therefore, node administrators must ensure that all ABOCC-recommended versions of the JDK are installed on their nodes and configured into Anetd.

The ABOCC may preconfigure Anetd to look for recommended JDK versions in particular locations. Node administrators then have the option of installing these recommended versions in the specified locations, or specifying the installation locations of their installed JDKs in the local configuration file (for each Anetd instance). The installation location for a particular version of the JDK is configured via a key of the form "`jdk<version>`", where version is the version number of the JDK, for example 1.2.2. Thus, a JDK configuration entry in the configuration file might appear as

```
jdk1.2.2     /path/to/jdk1.2.2
```

With the JDK locations so configured, EE developers can specify the use of a particular version via the "`V`" option in the `LOAD` command (see section 3.8).

Two additional keys, `java<version>` and `javag<version>`, specify the invocation strings for Java and Java with debugging support, respectively. The value of each key should be a list of strings, the first specifying the location of the Java binary, and the remaining specifying additional arguments with Java should be invoked, for example,

```
java1.2.2    "$(jdk1.2.2)/bin/java"
javag1.2.2   "$(jdk1.2.2)/bin/java", "-Xdebug", "-Djava.compiler=NONE",
             "-Xbootclasspath:$(jdk1.2.0)/jre/lib/rt.jar:$(jdk1.2.0)/lib/tools.jar",
             "-Xrunjdwp:transport=dt_socket,server=y,suspend=n", "-Xnoagent"
```

Along with each JDK configuration entry, there should be an entry to specify the location of the Java security policy file to be used with corresponding JDK. The location of a Java security policy file is configured via a key of the form `jsp<version>`, for example

Key	Description	Default value
user[a]	Owner of Anetd process	N/A
abone_role[a]	Acting ABone account	$(user)[b]
abocc_user[a]	Account acting in abocc role	abocc[cd]
		$(user)[ce]
hostname[a]	Local hostname	N/A
os[a]	Host OS type	N/A
home_dir	User home directory	(from passwd entry)
primary_role	ABone role of primary Anetd	anpub[d]
		$(user)[e]
port	Primary Anetd port	3322
child_port	First secondary Anetd port	8000
ip_addr	Listening IP address(es) (list)	0.0.0.0 (any)
debug_java	Allow Java EE debugging (yes/no)	no
debug_native	Allow native EE debugging (yes/no)	no
abocc_servers	ACL/TCL server URLs	(ABOCC defined)
halt_on_xcl_fail	Halt on ACL/TCL download failure	no
anetd_dir	Anetd top-level directory	$(home_dir)/anetd
bin_dir	Anetd binary directory	~$(abocc_user)/anetd/bin[d]
		$(anetd_dir)/bin[e]
code_dir	Local code directory	~$(abocc_user)/anetd/code/$(abone_role)[d]
		$(anetd_dir)/code[e]
conf_dir	Anetd configuration dir.	$(anetd_dir)/conf
abocc_conf_dir	ABOCC-owned conf. dir.	~$(abocc_user)/anetd/etc/$(abone_role)
conf_file	Master configuration file	$(abocc_conf_dir)/anetd.conf[d]
		$(conf_dir)/anetd.conf[e]
local_conf_file	Local configuration file	$(conf_dir)/anetd.conf.local
acl_server_dir	Master ACL server dir.	(ABOCC defined)
tcl_server_dir	Master TCL server dir.	(ABOCC defined)
acl_file	Master ACL file	$(conf_dir)/hosts.allow
tcl_file	Master TCL file	$(conf_dir)/web.allow
local_acl_file	Local ACL file	$(conf_dir)/hosts.allow.local
local_tcl_file	Local TCL file	$(conf_dir)/web.allow.local
run_dir	Anetd runtime directory	$(anetd_dir)/var
log_file	Anetd log	$(run_dir)/anetd.log
startup_file	Permanent command record	$(run_dir)/startup
jdk<version>	Location of JDK <version>	(ABOCC defined)
java<version>	JVM invocation (list)	(ABOCC defined)
javag<version>	JVM debugging invocation (list)	(ABOCC defined)
jlc<version>	Class loader for JDK <version>	abocc.LoadClient
jsp<version>	Security policy for JDK <version>	(ABOCC defined)
classpath	Java class path for EEs	$(bin_dir)
ld_library_path	Library path for EEs	$(bin_dir)
gdbserver_path	Location of gdbserver	gdbserver
accept_w_path	Gdbserver accept() wrapper	$(bin_dir)/accept.$(os).so

[a] This key is defined internally and its value may not be altered by configuration files.
[b] The default may be overridden via the -a command-line option.
[c] The default may be overridden via the -m command-line option.
[d] This value is the default when Anetd is running in an ABone configuration.
[e] This value is the default when Anetd is running standalone.

Table 1: Anetd configuration.

```
jsp1.2.2     $(abocc_conf_dir)/java-policy-1.2.2
```

Normally, the ABOCC will supply Java security policy files and specify their locations.

3.5 Anetd Invocation

Anetd is started on a network node as a user application. Anetd does not need any special privileges and can be started as described below. By default, Anetd puts itself in the background and redirects its standard output (stdout) and standard error (stderr) to files (as noted in the descriptions of the '-o' and '-e' options). Certain command-line options override corresponding configuration settings, as noted.

```
ad [-c conf_file] [-p port] [-u start_port] [[-i IP_address] ...]
   [-d anetd_dir] [-l log_file] [-o output_file] [-e error_file]
   [-v] [-f flag] [-q] [-g] [-a abone_role] [-m abocc_user]
   [-k] [-s] [-n] [-w seconds] [-h]
```

- **-c** specify the location of the master configuration file (overrides `anetd_conf`)
- **-p** specify the listening port (overrides `port`, default 3322)
- **-u** specify the listening port for the first child (overrides `child_port`, default 8000)
- **-i** specify an IP address to bind to (overrides `ip_addr`, default INADDR_ANY). This option may be used multiple times to specify a list of addresses to bind. Each Anetd process in a given ABone configuration (e.g., set of processes under different accounts) should specify exactly the same address list in exactly the same order.
- **-d** specify the root of the Anetd directory tree (overrides `anetd_dir`, default `~/anetd`). Use of this option may be helpful when running multiple Anetd configurations on the same host.
- **-l** specify the location of the command log (overrides `log_file`, default `~/anetd/var/anetd.log.<port>`).
- **-o** redirect Anetd's stdout to the specified file. By default, Anetd directs its stdout to `~/anetd.<pid>.out` when running as a daemon, and to the controlling terminal otherwise. Typically, Anetd writes very little to stdout.
- **-e** redirect Anetd's stderr to the specified file. By default, Anetd directs its stderr to `~/anetd.<pid>.err` when running as a daemon, and to the controlling terminal otherwise. Anetd writes most of its messages to stderr.
- **-v** be verbose
- **-f** set verbosity flag (time, pid, progname, funcname, level)
- **-q** be quiet
- **-g** allow remote debugging of EEs (overrides `debug_java` and `debug_native`)

- **-a** act in the specified ABone role. This option should be used only by experts who require nonstandard ABone node configurations.
- **-m** specify the OS account name of the Anetd process acting in the `abocc` role. This option should be used only by experts who require nonstandard ABone node configurations.
- **-k** keep running also if ACL file download failed (default)
- **-s** run in standalone mode
- **-n** do not run as a daemon; run in the foreground (used mainly for Anetd debugging)
- **-w** when starting a child process, hold for the specified number of seconds after fork(). This option is intended for use only by Anetd developers when debugging Anetd.
- **-h** print this help

The verbosity flags allow to customize the beginning of each line of output of Anetd:

- time: print the time
- pid: print the PID
- progname: print the program name
- funcname: print the function name
- level: print the diagnostic level

Eg:

```
ad.bsd44
This is anetd, release RELEASE_1_6_6

ad.bsd44 -f pid
[16370] This is anetd, release RELEASE_1_6_6

ad.bsd44 -f pid -f time
2000-05-15 16:28:30 GMT-07 [16372] main: This is anetd, release RELEASE_1_6_6
```

3.6 Signal Handling

Anetd catches the following signals:

- **SIGINT**: causes Anetd to perform a semi-orderly shutdown. Upon receipt of SIGINT, Anetd also attempts to terminate all running child processes (EEs) that have the autokill option set (the default). Anetd terminates child processes by sending them, in order, the signals SIGINT, SIGTERM, and SIGKILL until either the process terminates or each signal has been sent. See also the KILL command.

- **SIGHUP**: causes Anetd to reinitialize, including reloading of its configuration file(s) and refetching and reloading of all ACL and TCL files. Reinitialization does not include shutdown or restart of EEs.
- **SIGUSR1**: causes Anetd to reload its ACL and TCL files (master and local) without refetching the master files from the ACL/TCL server.

3.7 Client Application `sc`

Anetd's client application (sc) assists in issuing control commands formatted using ANEP and creating digital signatures for authentication. **sc** looks for the file $HOME/.anetd/id_secretkey and find the corresponding 512-bit private key associated with the clientid by the client. (if client doesn't input a client ID at the command line, sc will by default use the first private key it can find in the file $HOME/.anetd/id_secretkey).

After a valid 512-bit key is found, it creates a digital signature of the user command. The signature includes a timestamp to avoid replay attacks and is sent together with the command to the specified Anetd daemon. The **sc** command line usage is

```
sc [-q | -v] [-h] [-t timeout] [-p t|u] [-i client_id] [-f username]
   servername[:port] cmd args
```

- **-q** be quiet
- **-v** be verbose
- **-h** print this help
- **-t** specify the timeout for network operations (default 10 sec)
- **-p** specify transport protocol to use (t = TCP, u = UDP)
- **-i** specify which client ID to use
- **-f** forward to the secondary anetd with specified username
- **servername[:port]** hostname and port where anetd is listening (default 3322)
- <**cmd**> is one of the Anetd commands (see Section 3.8).
- **args** args to the control command <cmd>

3.8 Control Commands

Anetd control commands allow a client to deploy, configure and manage network application software.

3.8.1 Permanence

If one of the commands LOAD, CONF is prepended with a minus sign (`-`), the command becomes *permanent*[5], i.e. the command is recorded by anetd and will be automatically executed each time

[5] We may change the minus sign to something more self-evident in a future version.

anetd is restarted (this is analogous to a startup script).

A client (principal) must be specifically allowed in the ACL to issue permanent commands.

3.8.2 The LOAD/-LOAD command

```
[-]LOAD J=<jurl> | X=<url> | A=<url> | F=<url> [F=<url> ...]
        [T=<anep_id>] [N=<num>] [V=<jdk>] [S=<arg> ...] [E=<var:val> ...]
        [D=<dir>] [C=<description>] [I=<file>] [O=<file>] [R=<file>]
        [G=<debug_option>] [K=0|1|false|true|no|yes]
```

Instructs Anetd to download one or more files specified by URLs to a principal's working directory (or a subdirectory thereof), and optionally start a network service. **Note:** As of version 1.6, Anetd will permit the use of file URLs (`file:///`) only if authorized by the TCL (see also section 3.2.2).

See section 3.8.1 for the meaning of the optional `-`.

- **J=<jurl>** specifies a Java application.
  <**jurl**> is of the form `http://host[:port]/classpath/~class` or `file://[localhost]/classpath/~class` where

  - **host[:port]** specifies the server where the data is located following the normal URL conventions.
  - **classpath** is a path pointing to the base classpath of the Java application.
  - **class** is the class to invoke.

  Both **classpath** and **class** can be a series of directories separated by "/". For example, **J=http://abone-tcs.sdl.sri.com/java/ants-1.2/~ants/ConfigurationManager** specifies the URL of the ANTS application **ants.ConfigurationManager** located on SRI's http code server in the directory **java/ants-1.2/**.

- **X=<url>** means that <url> specifies a native binary executable.

- **A=<url>** means that <url> specifies an ANCORS thread. (see [2]).

- **F=<url>** means that <url> specifies a data file that should be simply transferred and stored in the local installation directory (see the D option for the meaning of installation directory in this context). The **F** option can be used independently or in conjunction with the **J** or **X** options. It is often used independently to preload certain files (such as configuration files) for use in subsequent invocations of EEs. Note that the **F** option may be used multiple times in a single command to load multiple files. When this option is used in conjunction with **J** or **X**, the specified EE is started only after all files specified in the command are successfully downloaded.

- **T=<anep_id>** is the ANEP ID of the service being deployed. If this argument is not present no packets will be demultiplexed to it by Anetd.

- **N=<num>** limits the maximum number of URL download attempts (using TCL- and DNS-equivalent rewrites[6]). Default is 0 (no limit). If <num> = 1, then Anetd will try only the exact URL specified on the LOAD command line.

- **V=<jdk>** tells Anetd to use the Java JVM version specified by **<jdk>**, eg V=jdk1.1.8. If this option is not present a default Java version will be used. If the version is only partially specified (e.g., jdk1.1), Anetd will use the first configured JDK whose version is prefix-matched by the requested version. See section 3.8.4 for information on how to determine which JDK versions are available on a node.

- **S=<arg>** tells Anetd to invoke the deployed service with **<arg>** as a command line argument. **<arg>** cannot contain any white space, so multiple arguments can be passed to the command line by using multiple instances of **S=<arg>**.

- **E=<variable>:<value>** tells Anetd to set the environment variable <variable> to the value <value>. Note that Anetd itself sets several environment variables. In most cases, the variables set by Anetd cannot be set effectively via the **E** option. The environment variables set by Anetd are as follows:

  - `CLASSPATH`: where Java may look for class files/libraries. This variable is set by Anetd to include first the local directory containing ABOCC class files (usually `~abocc/anetd/bin`) and second the EE working directory (including any subdirectory component specified by the **D** option, if any).
  - `LD_LIBRARY_PATH`: where the node may look for native shared libraries. This variable is currently set to the same value as `CLASSPATH`.
  - `DISPLAY`: the address of an X server display to be used by the EE for any windows it may open. If a value for DISPLAY is set via an **E** option, Anetd leaves that setting intact. Otherwise, Anetd sets `DISPLAY` to `"<client-IP-addr>:0.0"`, where `<client-IP-addr>` is the address of the host from which the LOAD command was issued. Note that the principal issuing the command is responsible for dealing with any X Window System authentication issues that may arise in the use of remote X servers.

  The remaining variables set by Anetd are listed below for completeness. They are intended for use primarily by ABOCC for administrative purposes.

  - `AD_DIR`: the Anetd installation directory
  - `AD`: the name of the Anetd executable
  - `AD_PID`: the Unix process ID of the Anetd process
  - `AD_COMMAND`: the full command line used to start the current Anetd process
  - `AD_START_DIR`: Anetd's working directory at the time the parent Anetd process was started
  - `AD_CWD`: Anetd's working directory at the time the child process was started
  - `AD_USER`: the OS account in which the Anetd process is running
  - `AD_ABONE_ROLE`: the ABone role in which the Anetd process is acting

[6]Starting with version 1.6, Anetd can use both TCL mirror lists and DNS mappings for code server redundancy. See also section 3.2.2.

- AD_ABOCC_USER: the OS account of the Anetd process that is acting in the `abocc` role.

- **D=<dir>** tells Anetd to use the directory <dir> as the root directory for the service installation. Anetd by default uses the directory **$(anetd_var)/<clientip>/** for installing all downloaded code; by specifying the **D=<dir>** option, Anetd installs all downloaded code in **$(anetd_var)/<clientip>/<dir>**.
- **I=<file>** redirects the standard input of the network service to the file <file> (to be created in the installation directory via the F option, otherwise this option is meaningless).
- **O=<file>** redirects the standard output of the network service to the file <file> (to be created in the installation directory).
- **R=<file>** redirects the standard error of the network service to the file <file> (to be created in the installation directory).
- **C=<description>** specifies a description <description> for the deployed services. This description is then returned to the client when **QUERY** commands are invoked. ABOCC requires this field to be a unique name for the EE, e.g., "ANTS" or "ANTSv1.3.1".
- **G=<debug_option>** specifies debug option for debugging EE remotely. The option can be either host:port or java. See Section 3.9 for more information.
- **K=0|1|false|true|no|yes** specifies whether the autokill option is set. When the autokill option is set, Anetd attempts to terminate the EE automatically when Anetd is shut down. If the autokill option is not set, Anetd will not attempt to terminate the EE automatically; however, it may still be signaled or terminated explicitly via the KILL command. The autokill option is set by default and can be overridden (K=0|false|no) only by a principal with sufficient privilege.

Example 3.2
For initiating an ASP EE on some node, the command is

```
sc.solaris host LOAD J=http://bro.isi.edu/ASP/~asp/AspMain
T=15 S=-d0 S=-l7 O=output R=error C=ASP_EE
```

For loading an ASP EE as a permanent EE on some node:

```
sc.solaris host -LOAD J=http://bro.isi.edu/ASP/~asp/AspMain
T=15 S=-d0 S=-l7 O=output R=error C=ASP_EE
```

Because the X and A types specify native executables, Anetd automatically appends the extensions **solaris, linux, and bsd44** to the URLs, depending on what platform Anetd is running on. For example, suppose that Anetd is running on a Linux machine; the URL `http://abone-tcs.sdl.sri.com/executables/ps` would actually fetch the file `http://abone-tcs.sdl.sri.com/executables/ps.linux`.

3.8.3 The RELOAD command

```
RELOAD anep <anep_id> | desc <description> | pid <pid>
```

Instructs Anetd to shutdown and restart a currently executing EE. The EE to be restarted can be specified by its ANEP type ID, its description (as specified by the 'C' option in the LOAD command), or its process ID (as reported by the QUERY command). An EE can be restarted only by its owner (the principal who originally issued the LOAD command) or by a privileged user. Regardless of who issues the RELOAD command, the EE is restarted as if the command had been issued by its owner. This ensures that the original owner will retain control over the EE after it is restarted.

Note that RELOAD can be used only on a running EE (one for which the QUERY command returns information). It cannot be used to restart a permanent EE that has terminated. To restart a permanent EE that has terminated, either Anetd must be restarted or the original LOAD command (without the leading '-') for that EE must be reissued.

Example 3.3
The following command could be used to restart the EE from Example 3.2 (assuming it is currently running):

```
sc RELOAD anep 15
```

Alternatively, the following command could be used:

```
sc RELOAD desc ASP_EE
```

3.8.4 The QUERY command

```
QUERY [ java | forward | demux | date | pid | conf | thread ]
```

Returns, to the client originating the command, a reply depending on the query.

- if no option is specified, returns a summary of the network services that were instantiated by Anetd.
- `java` returns a list of the available JVMs, distincted by version.
- `forward` returns a list of the secondary Anetds that have registered with the primary. This option is valid only for the primary daemon. A warning will be returned if the command "QUERY forward" is sent to a secondary daemon.
- `demux` returns a list of EEs registered with the primary daemon that have requested demultiplexing. This option is valid only for the primary daemon. A warning will be returned if the command "QUERY demux" is sent to a secondary daemon.
- `date` returns the system date and time.
- `pid` returns the PID of the Anetd process.

- conf [<prefix>] returns the current settings of all Anetd configuration keys matching the optionally specified prefix string. For each matching key, a line of output of the following form is generated:

```
<key_name> <value(s)> # [<flags>]
```

  where <key_name> is a matching configuration key, <value(s)> is the current comma-separated list of values assigned to the key, and <flags> is a collection of characters indicating whether certain internal handling flags are set for the key. The flags determine how future attempts to set values for the key will be handled:

  - P: the configuration key is *permanent*, i.e., its value is maintained upon reconfiguration
  - R: the configuration key is *read only*, i.e., its value set internally and cannot be changed by configuration files
  - A: additional values will be *appended* to the current list.

  A hyphen (-) indicates that the corresponding flag is not set. Note that flags are set and interpreted internally and cannot be changed by configuration files. They are included in the output primarily as an Anetd debugging aid.

  Note that the output is in Anetd configuration file format.

- thread [<item> ...] [<prefix>] returns selected information for each thread whose description field matches the optionally specified prefix. By default, all available information is returned for each matching thread. For a requesting principal without sufficient privilege, only information for threads owned by that principal is returned. One or more <item> keywords may be specified to select only certain information. The selected information is returned in the same order as the specified keywords. The available information items and their associated keywords are as follows:

  - owner: the principal ID of the thread owner
  - ip: the IP address from which the original LOAD command for the thread was issued
  - acmd: the Anetd command that started the thread
  - type: the thread type (Java, native, or ANCORS)
  - anep: the ANEP ID for the thread (for demultiplexing)
  - prog: for native threads, the name of the executable; for Java threads, the word "Java" followed by the Anetd key for the JDK version
  - base: the base URL for the thread's code, followed by the list of TCL-equivalent URLs used by Anetd to retrieve the code
  - class: the Java class portion of the URL requested by the LOAD command (Java threads only)
  - dir: the (node-local) working directory for the thread
  - files: additional files (if any) requested by LOAD command 'F' options
  - stdin: standard input redirection
  - stdout: standard input redirection

- `stderr`: standard input redirection
- `flags`: a space-separated list of flags set for the thread (`Permanent`, `Debug`, `Autokill`)
- `oscmd`: the operating system command used by Anetd to start the thread
- `env`: Environment variables set for the thread
- `pid`: the operating system process ID of the thread
- `start`: the (node-local) time the thread was started
- `uptime`: elapsed time since the thread was started
- `port`: the (node-local) port to which Anetd demultiplexes packets for the thread

Example 3.4 (QUERY)

```
1       $ sc d04 QUERY
2       text/ancors: anpub@d04.csl.sri.com:3322 RELEASE_1_6_6 bsd44
3       anetd_uptime 1:02:04:52
4       -----------------------------------------
5       130.107.19.127: "foo" ("anetd_ee.udp.963428600.bsd44", pid 5255,
...         permanent, anep_id 80, uptime 1:02:04:24,
...         working_dir "/home/anpub/anetd/var/7f136b82", autokill on)
6       -----------------------------------------
7       130.107.19.127: "<no description>" ("test-load-x.963522454.bsd44", pid 20390,
...         not permanent, anep_id 0, uptime 0:00:00:10,
...         working_dir "/home/anpub/anetd/var/7f136b82", autokill on)
```

Line 1 is the sc invocation.

Line 2 tells that Anetd is running on host `d04.csl.sri.com` at port `3322` as user `anpub`, that the operating system is `bsd44` (really FreeBSD) and that the release is `RELEASE_1_6`.

Line 3 tells that Anetd uptime is 1 day, 2 hours, 4 minutes.

Line 5 tells that a client with IP address `130.107.19.127` started the service `foo` whose program name is `anetd_ee.udp`, whose PID is `5255`, is `permanent`, has requested demultiplexing through ANEP ID `80`, has an uptime of 1 day and 2 hours, was started in the working directory (on the node) `/home/anpub/anetd/var/7f136b82`, and has the autokill flag set. Note that the description, program name, and directory are displayed as double-quoted strings. If these strings have embedded double-quotes, control characters, nonprintable characters, or other characters, such as “\”, that would normally need to be escaped, these characters will be displayed in C-style escaped form (e.g., “"” → “\"”, “\” → “\\”, newline → “\n”, etc.).

Line 7 describes another service, with some differences: it has no description, its program name is `test-load-x`, is not permanent, and hasn't asked for demultiplexing (its ANEP ID is 0).

Example 3.5 (QUERY java)

```
 1      sc.bsd44 sofia QUERY java
 2      text/ancors: molter@sofia.csl.sri.com:3322 HEAD_SNAP_2000_05_16 bsd44
 3      jdk1.1.8
 4      jdk1.2
```

Lines 1,2 are as in example 3.4.

Lines 3,4 tell the available JVM versions.

Example 3.6 (QUERY forward)

```
1   $ sc d04 QUERY forward
2   text/ancors: anpub@d04.csl.sri.com:3322 RELEASE_1_6 bsd44
3   host              user     anep_id   last_heard  created
4   127.0.0.1:8000    abocc    0         0:00:00:19  N/A
5   127.0.0.1:8001    anee1    1         0:00:00:17  N/A
6   127.0.0.1:8002    anee2    2         0:00:00:15  N/A
7   127.0.0.1:8003    anee3    3         0:00:00:13  N/A
8   127.0.0.1:8004    anee4    4         0:00:00:11  N/A
9   127.0.0.1:8005    anee5    5         0:00:00:09  N/A
```

Lines 1,2 are as in example 3.4.

Line 3 is a header.

Lines 4..9 are the secondary Anetds.

Example 3.7 (QUERY demux)

```
1   $ sc d04 QUERY demux
2   text/ancors: anpub@d04.csl.sri.com:3322 RELEASE_1_6_6 bsd44
3   host           user  anep_id demux_count last_demux registered
4   127.0.0.1:3721 anee1 80      4143        0:00:00:23 2000-07-12_12:03:20_GMT-07
```

Lines 1, 2, and 3 are as in example 3.6.

Line 4 is the EE of line 5 of example 3.4. The other EE of example 3.4 isn't listed here since it did not request demultiplexing. The field `demux_count` indicates the number of ANEP packets of the listed `anep_id` received by Anetd since the EE was registered. The field `last_demux` shows the time elapsed since the last such packet was received. If no such packets have been received, "`N/A`" is displayed. The field `registered` shows the time at which the EE was registered.

Example 3.8 (QUERY conf)

```
1   $ sc d04 QUERY conf anetd_dir
2   text/ancors: anpub@d04.csl.sri.com:3322 RELEASE_1_6_6 bsd44
3   anetd_dir "/home/anpub/anetd" # [---]
```

Lines 1 and 2 are as in example 3.6.

Line 3 shows the setting for the requested configuration key, `anetd_dir`. It has no flags set, and its current value is shown to be `/home/anpub/anetd`.

Example 3.9 (QUERY thread)

```
1  $ sc -f ahost d04 QUERY thread
2  text/ancors: anee1@ahost:3322->8001 RELEASE_1_6_7 bsd44
3  ----------
4  Test_EE
5  Owner: auser@adomain
```

```
 6  Source IP: 192.168.100.101
 7  Anetd command: LOAD X=http://abone-tcs.sdl.sri.com/test/testee
    S=testfile T=55 F=http://abone-tcs.sdl.sri.com/test/testfile
 8  Type: native
 9  ANEP ID: 55
10  Program: /home/anee1/anetd/var/6564a8c0/testee.985720883.bsd44
11  Base URL: http://abone-tcs.sdl.sri.com/test/testee.bsd44
12  Equivalent URLs: http://abone-tcs.sdl.sri.com:80/test/testee.bsd44,
    http://205.178.57.134:80/test/testee.bsd44
13  Java class: <NULL>
14  Directory: /home/anee1/anetd/var/6564a8c0
15  Additional URL 1: http://abone-tcs.sdl.sri.com/test/testfile
16  Equivalent URLs 1: http://abone-tcs.sdl.sri.com:80/test/testfile,
    http://205.178.57.134:80/test/testfile
17  Local file 1: /home/anee1/anetd/var/6564a8c0/testfile
18  Stdin: <NULL>
19  Stdout: <NULL>
20  Stderr: <NULL>
21  Flags: Autokill
22  OS command: /home/anee1/anetd/var/6564a8c0/testee.985717281.bsd44 testfile
23  Env 0: AD_DIR=/home/anee1/anetd
24  Env 1: AD=ad.bsd44
25  Env 2: AD_PID=7932
26  Env 3: AD_COMMAND=/home/abocc/anetd/ad.bsd44
27  Env 4: AD_START_DIR=/home/abocc
28  Env 5: AD_CWD=/home/abocc
29  Env 6: AD_USER=anee1
30  Env 7: AD_ABONE_ROLE=anee1
31  Env 8: AD_ABOCC_USER=abocc
32  Env 9: DISPLAY=192.168.100.101:0.0
33  Env 10: CLASSPATH=/home/abocc/anetd/bin:/home/anee1/anetd/var/6564a8c0
34  Env 11: LD_LIBRARY_PATH=/home/abocc/anetd/bin:/home/anee1/anetd/var/6564a8c0
35  PID: 8021
36  Start: 2001-03-27_10:21:21_GMT-08
37  Uptime: 0:00:03:21
38  Port: 8017
```

Lines 1 and 2 are as in the earlier examples. Line 3 is simply a separator that is output at the beginning of the information for each thread. Lines 4 through 38 show all available information for the one running thread owned by the issuer of the query. The first line (line 4) is the thread description. Each remaining line begins with a short description of the information it contains.

3.8.5 The KILL command

`KILL [-<sigspec>] <pid> [force]`

Allows a client to signal or terminate an EE.

- **<sigspec>** is the (symbolic or numeric) Unix signal to be sent to the EE (note that the '-' prefix is required). If not specified, Anetd attempts to terminate the EE by sending it, in order, the signals SIGINT, SIGTERM, and SIGKILL until either the thread terminates or each signal has been sent.
- **<pid>** is the PID of the thread to be terminated by Anetd.

The <pid> value should be retrieved by using the QUERY command. Only the principal that started the EE, or a principal with the necessary privilege, may signal or terminate an EE. When a thread terminates, Anetd will automatically garbage-collect all resources allocated to the terminated service.

As of version 1.6.7, users with administrative privilege may specify the PID of the Anetd process to have Anetd signal itself. In this case, Anetd first checks whether the signal is likely to cause it to terminate. If so, it issues a warning to that effect and does not send the signal. Such signals may be forced without warning by appending the keyword "`force`" to the command. See also the RAISE command.

Example 3.10

```
$ sc d04 QUERY
text/ancors: anpub@d04.csl.sri.com:3322 RELEASE_1_6_6 bsd44
anetd_uptime 1:02:42:01
------------------------------------------
130.107.19.104: "<no description>" ("anetd_ee.udp.963522454.bsd44", pid 20390,
    not permanent, anep_id 0, uptime 0:00:37:19,
    working_dir "/home/anpub/anetd/var/68136b82", autokill on)

$ sc d04 KILL 20390
text/ancors: anpub@d04.csl.sri.com:3322 RELEASE_1_6_6 bsd44
KILL: pid 20390 terminated
```

Example 3.11

```
$ sc d02 QUERY
text/ancors: anpub@d02.csl.sri.com:3322 RELEASE_1_6_6 bsd44
anetd_uptime 1:17:33:29
------------------------------------------
130.107.19.102: "<no description>" ("anetd_ee.udp.963522454.bsd44", pid 17134,
    not permanent, anep_id 0, uptime 0:00:37:19,
    working_dir "/home/anpub/anetd/var/66136b82", autokill on)

$ sc d02 KILL -TERM 17134
text/ancors: anpub@d02.csl.sri.com:3322 RELEASE_1_6_6 bsd44
KILL: pid 17134, sent signal 15 (Terminated)
```

3.8.6 The GET and TGET commands

```
GET <relative_path/file>
```

```
TGET <relative_path/file>
```

Allows a client to retrieve a file through Anetd.

- **<file>** specifies the name of the file to be retrieved

Anetd by default uses the directory \$(anetd_var)/<clientip> to look for the file `<relative_path/file>`. Anetd only allows the client that originally created the file <file> to retrieve it. The content of the file is returned to the client in the acknowledge message. The file returned cannot be greater than approximately 64K bytes, since this is the maximum payload of an ANEP packet, and we are using ANEP as datagram. Larger files are truncated to their maximum allowed size. The GET command returns (at most) the first 64K of the file, while the TGET command returns (at most) the last 64K of the file.

Example 3.12

```
sc.solaris host GET output
```

3.8.7 The PUT command

```
PUT <remote_relative_path/remote_filename> <local_path/local_filename>
```

Allows a client to upload a file through Anetd. The maximum file size is 63K bytes. The content in the file (local_filename) will be put in the file (remote_filename) in the client's code installation directory at the server side.

- **<remote_relative_path/remote_filename>** specifies the name of file to be created. It will be written under the code installation directory at the host side. For security reasons, it cannot be written under any parent directory of the installation directory, but can be written under any child directory specified by user as remote_relative_path.
- **<local_path/local_filename>** specifies the path of file and the name of the local file to be uploaded

Example 3.13
This command uploads the file /homes/user1/config at the client side (apple) to the file /homes/user/anetd/var/88aa22bb/newdir/testfile on host side (berry).

```
[berry]% ad.bsd44
(machine berry has ip address in hex as 88aa22bb)

[apple]% sc.solaris host PUT newdir/testfile /homes/user1/config
```

3.8.8 The GETCOMM/-GETCOMM Command

```
[-]GETCOMM [noquery]
```

Return, to the client originating the command, a list of the commands most recently executed by Anetd since it was last started (GETCOMM) or a list of the permanent commands (-GETCOMM). The length of the returned list is currently limited to approximately 250 entries.

If the `noquery` option is specified (version 1.6.2 or later), QUERY commands are omitted from the list.

The list has the following format:
<index> <time> <clientip> <command>

- <**index**> is an index generated by Anetd (0,1,2 etc.)
- <**time**> is when the command was executed (day, date, time)
- <**clientip**> is the IP address in hex of the client that installed the service in hexadecimal format
- <**command**> is the command that was sent

Example 3.14

```
$ sc d04 GETCOMM
text/ancors: anpub@d04.csl.sri.com:3322 RELEASE_1_6_2 bsd44
0 Wed Jul 12 11:56:11 2000 7f136b82 QUERY version 1 molter@sdl.sri.com forward
```

3.8.9 The GETACL Command

`GETACL [ master | local | all ]`

Displays the Acess Control List (ACL) to the client. If `master` is specified, only the master ACL is returned. If `local` is specified, only the local ACL is returned. If `all` is specified, both ACLs are returned. The default is `master`.

Example 3.15

```
sc.solaris host GETACL all
```

3.8.10 The GETTCL Command

`GETTCL [ master | local | all ]`

Displays the Trusted Codeserver List (TCL) to the client. If `master` is specified, only the master TCL is returned. If `local` is specified, only the local TCL is returned. If `all` is specified, both TCLs are returned. The default is `master`.

This command was formerly known as GETWEB, whose use is now deprecated.

Example 3.16

```
sc.solaris host GETTCL
```

3.8.11 The CONF/-CONF Command

`[-]CONF <handle> <symbol> [<args>]`

Applies to ANCORS threads and is similar to a remote procedure call.

See Section 3.8.1 for the optional [-].

- `<handle>` shared library handle, obtained by a previous `LOAD A`.
- `<symbol>` name of the function to invoke, to be found in the shared library identified by `<handle>`.
- `[<args>]` command line arguments to be passed to function `<symbol>`.

Example 3.17

```
$ sc host CONF 2352135 ospf_monitor_start arg1 arg2 arg3
```

3.8.12 The RMPERM Command

`RMPERM <index>`

Allows a client to remove a permanent service from Anetd's restart script.

- **<index>** is the index of the service to be terminated by Anetd (0,1,2 etc.). This value should be retrieved by using the -GETCOMM command.

Example 3.18

```
sc.solaris host RMPERM 0
```

3.8.13 The RAISE Command

`RAISE -<signal> [force]`

Causes Anetd to send itself the specified signal. This command may be used only by principals with ABone administrative privilege. If the keyword "`force`" is not specified, Anetd first checks whether the specified signal is likely to cause Anetd to terminate, and if so, it returns a warning and does not raise the signal. See also the KILL command and Section 3.6.

3.8.14 The RECONF Command

`RECONF`

Causes Anetd to reload its configuration files, including downloading and reloading the master ACL and TCL files. This command may be used only by principals with ABone administrative privilege, and is equivalent to the command `RAISE -HUP`. See also Section 3.6.

3.8.15 The REAUTH Command

`REAUTH`

Causes Anetd to reload, but not refetch, its ACL and TCL files. This command may be used only by principals with ABone administrative privilege, and is equivalent to the command `RAISE -USR1`. See also Section 3.6.

3.8.16 The RESTART Command

`RESTART [exec]`

Checks whether Anetd can be self-restarted and optionally causes Anetd to shutdown and restart. The RESTART command with no arguments simply checks whether Anetd is likely to succeed in attempting a self-restart. If the keyword "`exec`" is specified, Anetd will attempt a self-restart if it appears safe to do so. The RESTART command may be used only be principals with ABone administrative privilege.

The effect of a successful self-restart is essentially the same as if Anetd had been sent the signal SIGINT (see Section 3.6) and restarted locally on the node. In particular, EEs with the autokill flag set (the default) are terminated by Anetd when it shuts down, and only permanent EEs are restarted when Anetd restarts.

3.9 EE Debugging

Note that, to be able to use the debugging facilities, you must explicitly tell Anetd to allow debugging with the **-g** switch.

3.9.1 Start a Java EE under jdb

To start a Java EE under jdb, use the **G=java** option on the Anetd LOAD command. Depending on the version of the JDK used, the remote JVM will display a line of the form

```
Agent password=<passwd>
```

or

```
Listening for transport dt_socket at address: <port>
```

This output is captured by Anetd and returned to the client. The client user must then use this information to connect a local JDB process to the remote JVM[7].

Example 3.19
Assuming that `remote.host.domain` is running Anetd with remote debugging enabled, the following commands might be used to establish a remote Java debugging session:

```
[local]% sc remote.host.domain LOAD J=http://bro.isi.edu/ASP/~asp/AspMain
         T=15 S=-d0 S=-17 O=output R=error C=ASP_EE V=jdk1.1.8 G=java
Agent password=3upmuk

[local]% jdb -host remote.host.domain -password 3upmuk
```

or, with a more recent JDK:

[7]The client must use `jdb` from a JDK version compatible with, and ideally the same as, the remote JVM.

```
[local]% sc remote.host.domain LOAD J=http://bro.isi.edu/ASP/~asp/AspMain
         T=15 S=-d0 S=-17 O=output R=error C=ASP_EE V=jdk1.2.2 G=java
Listening for transport dt_socket at address: 1742

[local]% jdb -attach remote.host.domain:1742
```

3.9.2 Start an EE under gdb

To start an EE under gdb, use the **G=<host>:<port>** option with the Anetd LOAD command. Here <host> is the name of the *client host* and <port> is an available TCP port on the *target node*. This will cause Anetd to start the EE under a variant of gdb called **gdbserver**. Gdbserver will listen for a TCP connection to the specified port. The developer can then execute gdb and issue the command **'target <node>:<port>'** inside gdb to cause it to open the specified TCP connection to the gdbserver instance on the target node. Debugging can proceed as if the EE were local to the client node. Note that these procedures require the source code only on the EE developer's client node.

The <host> portion is currently ignored by gdbserver. That is, gdbserver will not check whether the client IP address is allowed. For now, <host> is just a dummy parameter. However, to avoid introducing a security hole, the server needs to make sure that remote gdb debugging permission is given only to the client who issued the LOAD command that started the EE. To avoid changing gdbserver source code, we added a **shared library wrapper** around the accept system call, which checks whether the remote IP address matches. If so, the connection is allowed; otherwise it is dropped.

Example 3.20

```
[berry]% ad.solaris -g

[apple]% sc.solaris berry:3322 LOAD X=http://sequoia.csl.sri.com:7000/test/server
T=52 S=hosts G=apple:4000
```

4 Conclusion

The Anetd home page is `http://www.sdl.sri.com/projects/activate/anetd/`, where it can be downloaded free of charge. Source code is available since version 1.5, subject to the terms of the Anetd Community Software License, available from the same site.

The ABone homepage is `http://www.isi.edu/abone/`.

Anetd runs out-of-the-box on recent versions of FreeBSD, Linux and Solaris.

Please report porting problems and any bugs or inconsistency you found running Anetd to `abocc@isi.edu`).

Please also notify us of any interesting uses of Anetd in managing networked systems. (`abocc@isi.edu`)

4.1 Credits

Anetd includes software developed by

- the University of Southern California, Information Sciences Institute (ISI).
- the Royal Institute of Technology, Stockholm, Sweden.
- Todd C. Miller of the OpenBSD project.

Marco Molteni would like to thank Jeff Kann at ISI, for his ability at finding always new bugs ;-).

References

[1] Scott Alexander, Carl A. Gunter, Angelos D. Keromytis, Gary Minden, and David Wetherall. ANEP: Active network encapsulation protocol. *http://www.cis.upenn.edu/ switchware/ANEP/docs/ANEP.txt*, 1997.

[2] Livio Ricciulli, Phillip Porras, and Nachum Shacham. ANCORS: Adaptable network control and reporting system. *SRI International Technical Report SRI-CSL-98-01*, 1998.

[3] D. J. Wetherall, J. V. Guttag, and D. L. Tennenhouse. Ants: A toolkit for building and dynamically deploying network protocols. *Submitted to IEEES OPENARCH'98*, 1998.